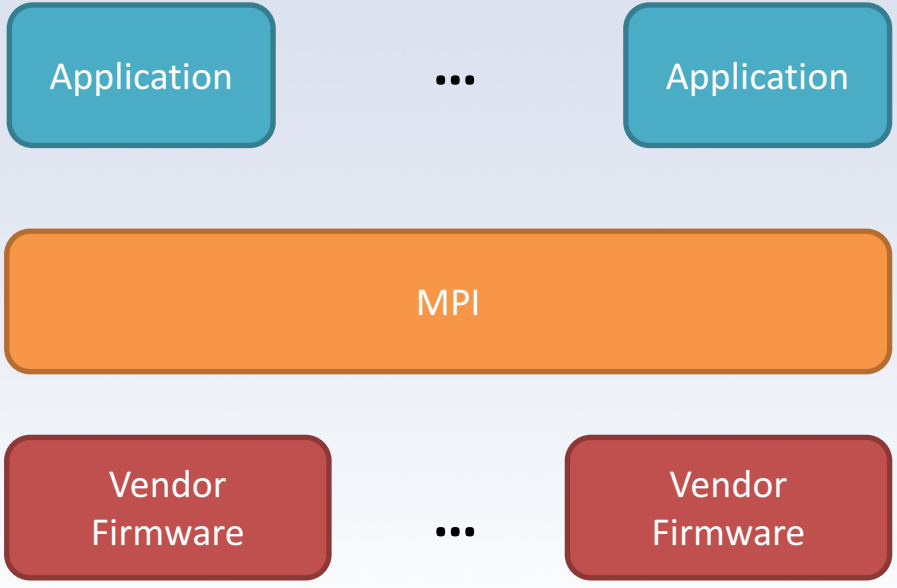
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

MPI is too High-Level
MPI is too Low-Level

Marc Snir

illinois.edu

"High-Level" MPI



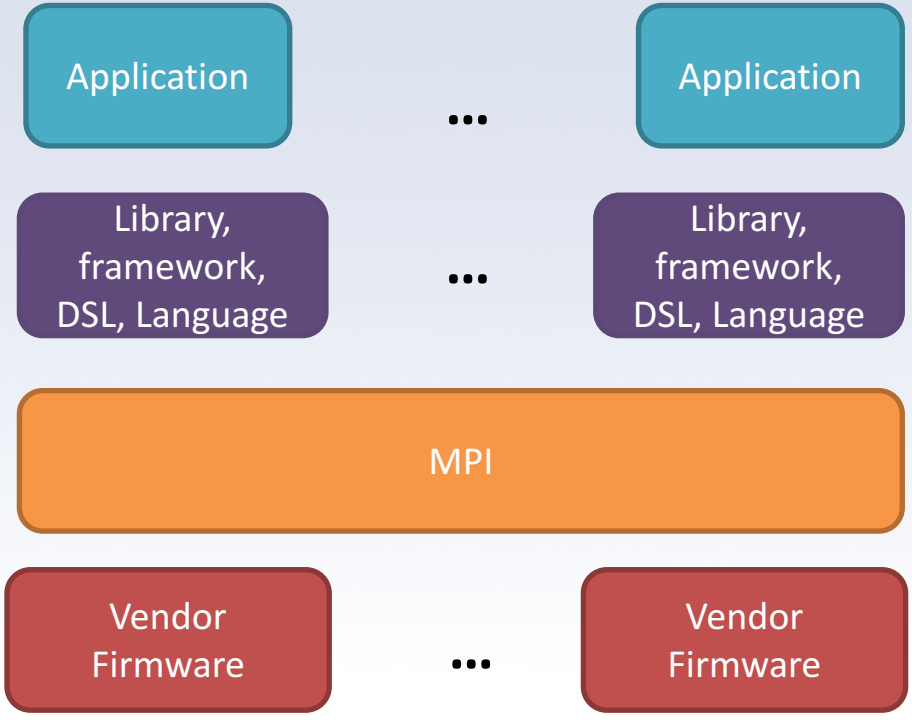
MPI is an Application Programming Interface

- from MPI-1.0:``Design an application programming interface (not necessarily for compilers or a system implementation library)."
- *Claim: MPI is too low-level for this role*

MPI is too Low Level

- Critique is (almost) as old as MPI: MPI is bad for programmer productivity
- Recent example (2015):
  - *HPC is dying and MPI is killing it* (Jonathan Dursi)
- "MPI is the assembly language of parallel programming"
  - Not used as a compliment…
- Largely irrelevant: Most "use" of MPI is indirect

"Low-Level" MPI



MPI is a communication run-time that is not exposed to applications

- In the back of our mind during MPI design
  - But this view did not influence MPI design
- *MPI is too high-level for this role*

MPI is too High Level

- An **assembly** is a low-level programming language … in which there is a very strong correspondence between the language and the architecture's machine code instructions. (Wikipedia)
- *MPI is not "the assembly language of parallel programming"*
- There is a large semantic gap between the functionality of a modern NIC and MPI
  - MPI has significant added functionality that necessitates a thick software stack
  - MPI misses functionality that is provided by modern NICs

Trivial Example: Datatypes (1)

- Many frameworks/DSL's have their own serialization/deserialization capabilities
  - These will be optimized for the specific data structures used by the framework (trapezoidal submatrices, compressed sparse matrices, graphs, etc.)
- For static types, the serialization code can be compiled – this is much more efficient than MPI interpretation of a datatype
- Some early concerns about heterogeneity (big/small endian, 32/64 bits) are now moot

Trivial Example: Datatype (2)

- High-level MPI needs datatypes (or templated functions?)
- Low level MPI needs transfer of contiguous bytes

- *Why care, you have both in MPI?*
  1. Each extra argument and extra opaque object is extra overhead
  2. Large, unoptimized subsets of MPI are deadweight that slow development

(1) Simple most communication call

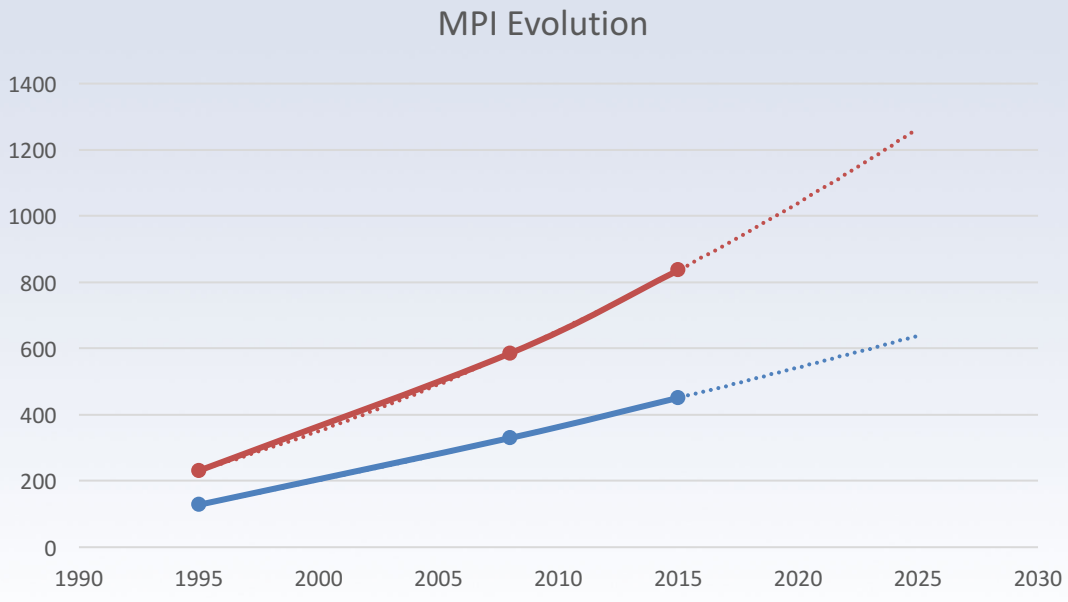
- **int MPI_Irecv(void** **buf*, **int** *count*, **MPI_Datatype** *datatype*, **int** *source*, **int** *tag*, **MPI_Comm** *comm*, **MPI_Request** **request* **);**

- Three opaque objects (indirection)
- Two arguments have "special values" (branches)
- Communication can use different protocols, according to source (shared memory or NIC)
- An API should have reasonable error checking

- None of that is needed in a low-level runtime

(2) MPI Evolution

- MPI 1.1 (June 1995)
  - 128 functions, 231 pages
- MPI 2.1 (June 2008)
  - 330 functions, 586 pages
- MPI 3.1 (June 2015)
  - 451 functions, 836 pages



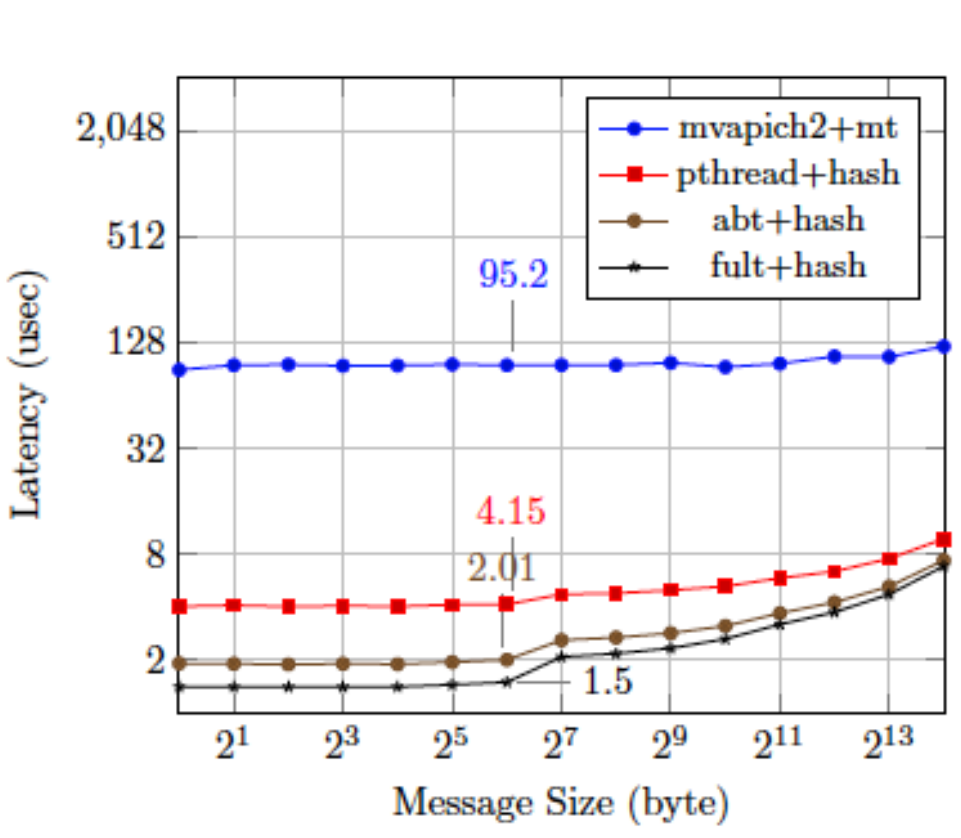
Continued growth at current rate is not tenable!

Problems of Large MPI

- Hard to get a consistent standard
  - E.g., fault tolerance
- Hard to evolve ~ 1 MLOC code
- Most features are not used, hence not optimized, hence not used – vicious circle

Simple Example: Don't Cares & Order

- Don't cares and ordering constraints prevent efficient implementation of MPI_THREAD_MULTIPLE
  - Problem is inherent to MPI's semantics
  - Getting worse with increased concurrency
  - Good support for MPI_THREAD_MULTIPLE is possible with no dontcares and is essential to future performance



H V Dang, M Snir, B Gropp

MPI Solutions

High-Level MPI

- Provide mechanism to indicate no order or no don't-care on communicator
  - Yet another expansion of standard
  - Slowdown because of an extra branch
  - Difficulty of using two fundamentally different matching mechanisms

Low-Level MPI

- Get rid of message ordering
  - Usually not needed; if needed, can be imposed at higher-level with sequence numbers
- Use a "send don't care" to be matched by a "receive don't care"
  - Assume sender "knows" the receiver uses dontcare.

Complex Example: Synchronization

- Point-to-point communication:
  - Transfers data from one address space to another
  - Signals the transfer is complete (at source and at destination)
- MPI signal = set request opaque object
- Problems:
  - Forces application to poll
  - Provides inefficient support to many important signaling mechanisms

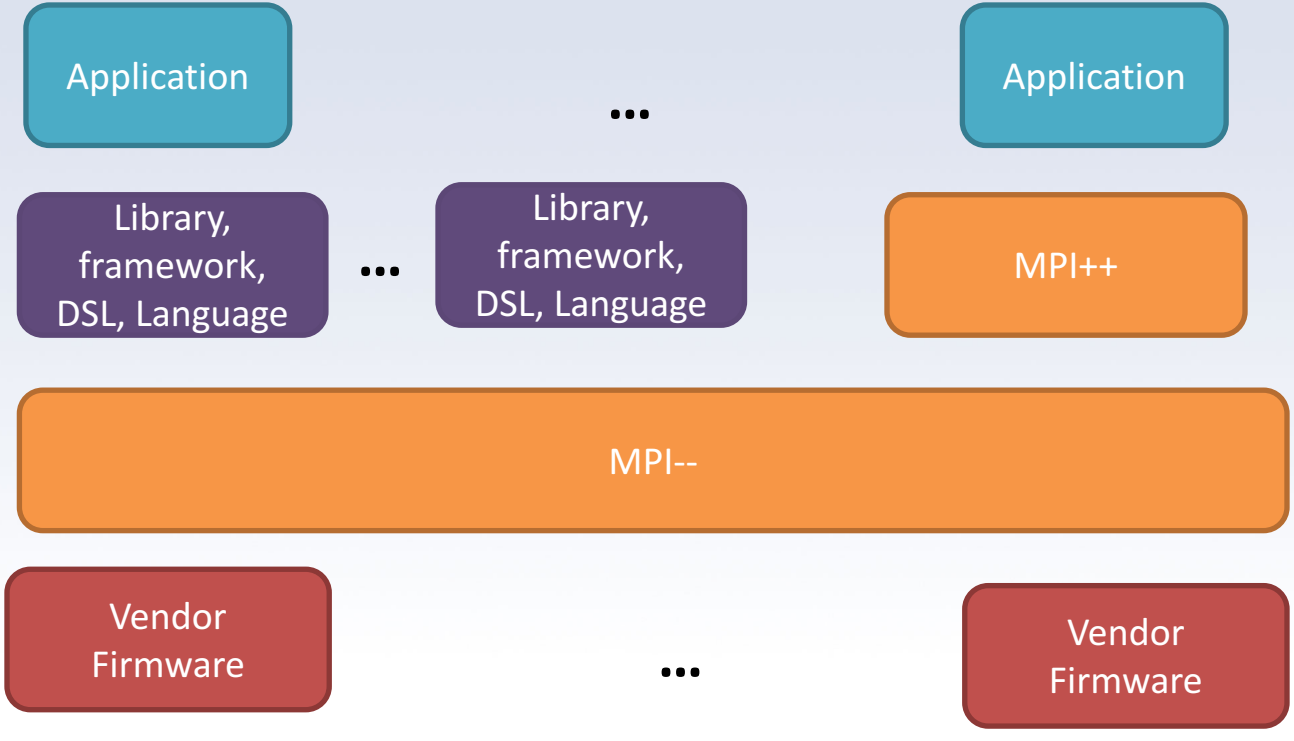
Signaling Mechanisms

1. Set flag
2. Decrement counter
3. Enqueue data + metadata in completion queue
4. Enqueue metadata + ptr to data in completion queue
5. Wake up (light-weight) thread
6. Execute (simple) task – active message
7. fence/barrier

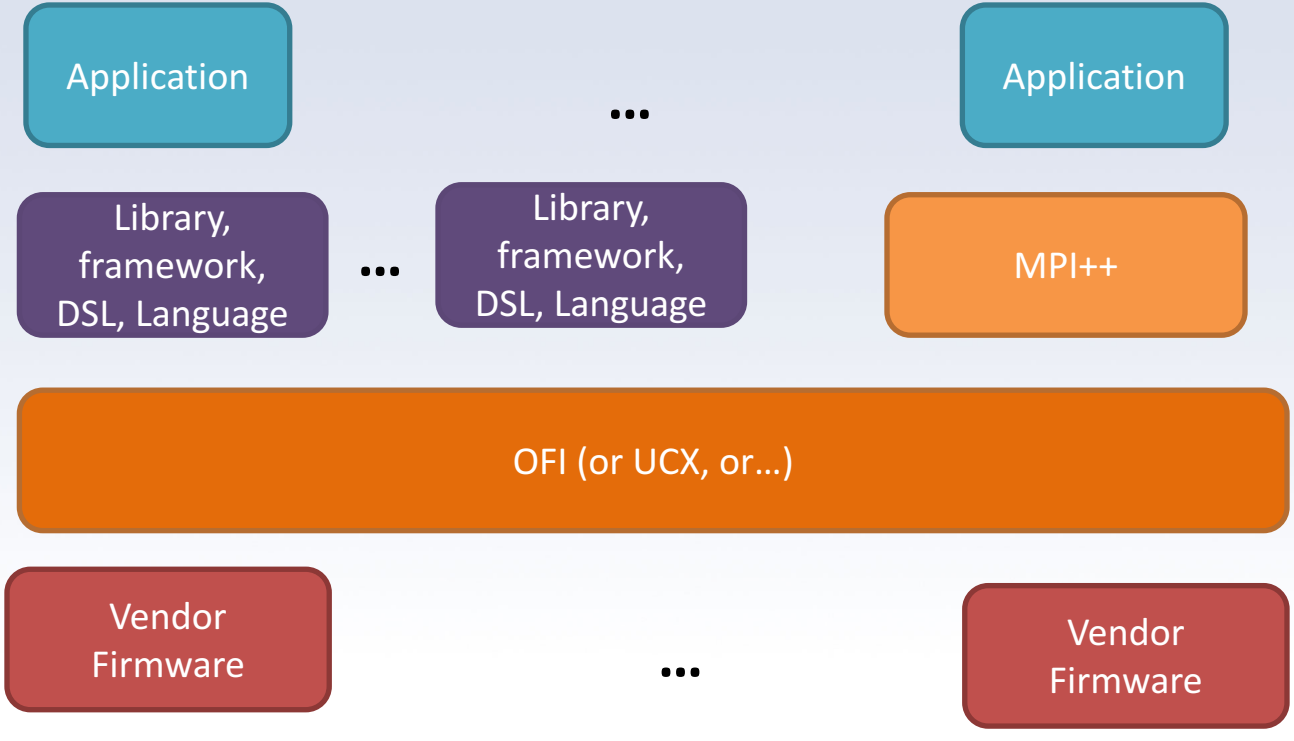
Signaling Mechanisms

- Each of these mechanisms is used by some framework
- All are currently implemented (inefficiently) atop MPI by adding a polling communication server
- 1-4 & 7 can be easily implemented by NIC (many are already implemented)
- 5 could be implemented by NIC if comm. library and thread scheduler agree on simple signaling mechanism (e.g., set flag)
- 6 can be implemented in comm. library (callback) with suitable restrictions on active message task (OK at low level interface)

Should we Bifurcate MPI?



Do we Need to Invent Something New?



Not sure

- Will industry converge to one standard without community push?
  - Standards are good, so we need many…
- Need richer set of "completion services" than currently available in OFI (queues and counters)
  - *Need more help from NIC and library in demultiplexing communications*
- Need (weak) QoS & isolation provisions in support of multiple clients


Questions?